# Models for Sequential Data

Machine Learning 2020
mlvu.github.io

## the plan

**part 1:**
Sequences

Markov models

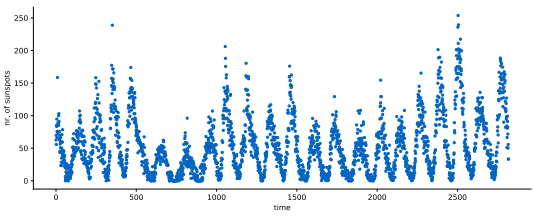
Embedding models
Word2Vec

**part 2:**
Recurrent Neural Nets

LSTMs



## numeric 1-dimensional



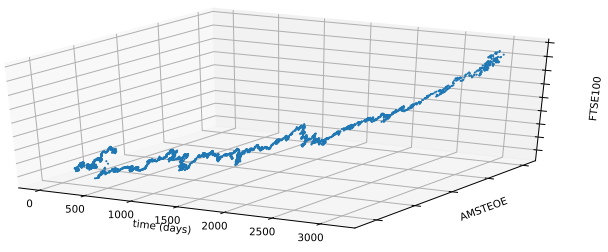



Before we start looking at different models to learn from sequential data, it pays to think a little bit about the different types of sequential datasets we might encounter.

As with the traditional setting (a table of independently sampled instances) we can divide our features into numeric and discrete.

A single 1D sequence might look like this. We could this of a stock price over time, traffic to a webserver, or atmospheric pressure over Amsterdam. In this case, the data shows the number of sunspots observed over time.

## numeric n-dimensional






Sequential numeric data can also be multidimensional. In this case, we see the closing index of the AEX and the FTSE100 over time. This data is a sequence of 2D vectors.

## symbolic (categorical) 1-dimensional

the, cat, sat, on, the, mat

t, h, e, _, c, a, t, _, s, a, t, _, o, n, _, t, h, e, _, m, a, t

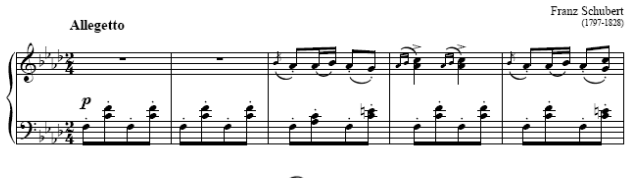
5

If the elements of our data are discrete (analogous to a categorical feature), it becomes a sequence of symbols. Language is a prime example. In fact, we can model language as a sequence in two different ways: as a sequence of words, or as a sequence of characters.

---

## symbolic n-dimensional

the/ART cat/NOUN sat/VERB on/PREP
the/ART mat/NOUN




6

We can also encounter sequences with multiple categorical features per timestamp, like music, or tagged language. These can often be difficult to represent.

---

## single sequence vs set-of-sequences

Dear recipient, Avangar Technologies announces the beginning of a new unprecendented global employment campaign
...
reviser yeller winers butchery twenties
Due to company's exploding growth Avangar is expanding business to the European region.
During last employment campaign over 1500 people worldwide took part in Avangar's business
and more than half of them are currently employed by the company. And now we are offering you
**spam**

BENCHMARK SUPPLY 7540 BRIDGEGATE COURT
ATLANTA GA 30350 ***LASER PRINTER TONER CARTRIDGES*** ***FAX AND COPIER TONER***
CHECK OUT OUR NEW CARTRIDGE PRICES :
**spam**

On 08/08/02 15:26 -0700, Chip Paswater wrote:
| > Well, a little more than one bit -- you have to transmit the signature, plus now the entire ... ID.
| > Plus the rights to all intellectual property contained in any
| > email message you submit (I guess technically there are no bits
| > in the rights assignment)
|
**ham**

If you own a travel related website, why not submit your site to our directory. Just select the appropriate category and subcategory and enter your title ... description.

Click here to start: http://www.holprop-travel-directory.com
**spam**

| hmmm, I assume you're going to report this to the nmh folks? Yes, I will, sometime, after I look at the nmh sources
and see what they have managed to break, ...why.

But we really want exmh to operate with all the versions of nmh that
exist, don't we? The patch to have exmh do the right thing, whether this
bug exists, or not, is trivial, so I'd suggest including it.
**ham**

Hello I am emailing you as we found your holiday property at [Source] and would love you to list it on our website for free:

http://www.holidayrentals.org

This web site now has over 1000 holiday rentals listed in less than 2 months, so
**spam**

Hello again self-catering.co.uk was recently launched and if you have not yet added your property for free then please do as we have had many properties added in the past few days.
**spam**

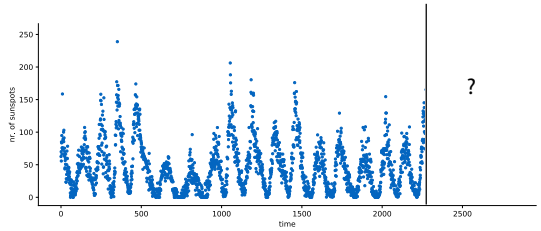
7

Then there is the question of what we're trying to predict. Do we have one sequence per instance, and are trying to, for instance, classify the sequences? That's what we see here: each email is a sequence (of words or or characters).

The instances themselves do not have a very sequential ordering (emails do have a timestamp, but this ordering is usually ignored).

---

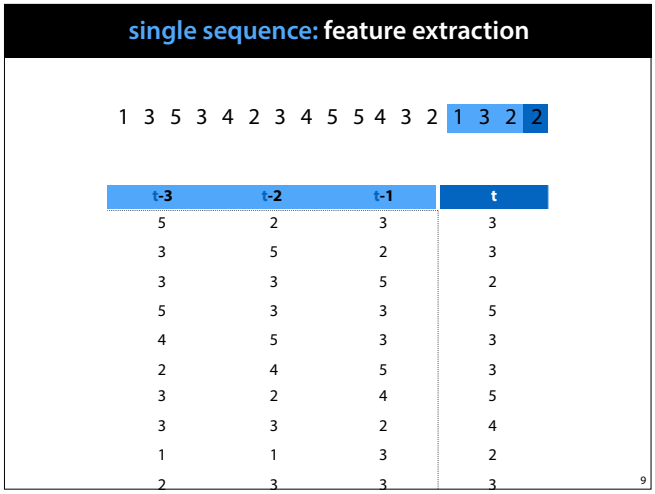
## single sequence




8

An entirely different setting is one where the dataset as a whole is a sequence, and the instances are ordered.

In that case, we often want to predict the future values of the sequence based on what we've seen in the past. To keep things simple, let's stick with 1D numeric data, like our sunspots example.




One simple way to achieve this, is to translate it to a classic regression problem by representing each point by the m values before it. This gives us a table with a target label (the value at time t) and m features (the m preceding values). If we successfully train a regression model on this task, we can then use it to predict future values of the sequence.

Many other features are possible: mean over the whole history. Mean over the last 10 points, variance one the last 10 points, etc.

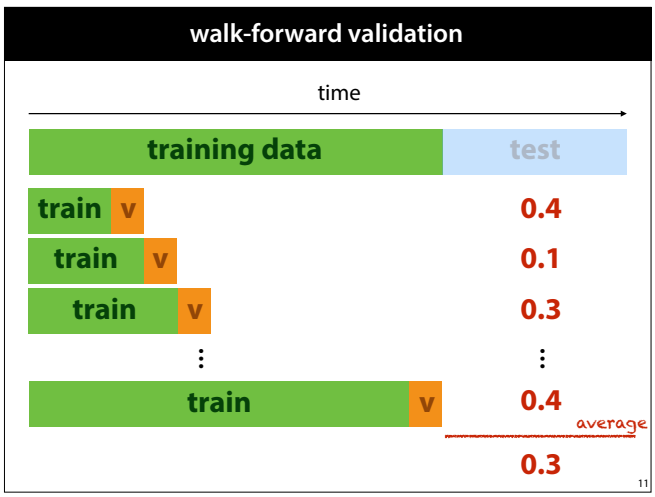
But, there's a problem. If we shuffle this data, and pick a test and training set, the classifier is trained on data that happens in the future to data it will be tested on!

---


Whenever you have questions about how to approach something like this, it's best to think about the real world setting where you might apply your trained model.

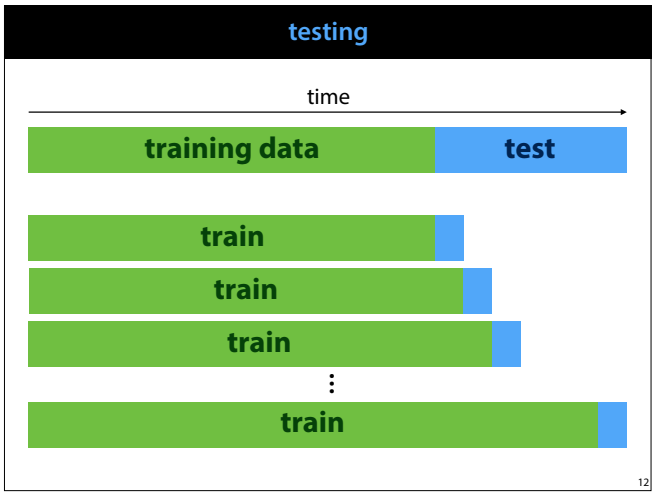
In this case, we are going to apply the prediction model on the three most recent values, to predict the next value.The test data should simulate that situation. That means that the training data should not contain instances that occur later than the test data.

---




To check your algorithm, to do cross validation.

---




Note, only if the target labels have a meaningful ordering in time. In spam classification, for instance, this is not necessary (emails do have a timestamp, but seeing an email from the future does not usually give you an unrealistic advantage).

## summary: sequential data

Sequences: consisting of numbers, vectors or symbols

Dataset: consisting of a **sequence per instance**, or a **sequence of instances**.
For a sequence of instances, carful with your test and validation

Both can be converted to fit classic machine learning through feature extraction.

Rest of the lecture: **1D symbolic sequences**.
Extension to nD and/or numeric is often trivial

13

But, often it's better to take a model that can consume sequences natively.

---

## modeling sequences: probability

$$\begin{aligned}&\mathrm{p}(\text{“congratulations you have won a prize”})\\ &= \mathrm{p}(W_1{=}\text{congratulations}, W_2{=}\text{you}, W_3{=}\text{have},\\ &\quad W_4{=}\text{won}, W_5{=}\text{a}, W_6{=}\text{prize})\end{aligned}$$

$$\mathrm{p}(W_1, W_2, W_3, W_4, W_5, W_6)$$

14

We'll start by looking at some ways to model sequences in probabilistic terms.

When modelling probability, we usually break the sequence up into its **tokens** (in this case the words of the sentence) and model each as a random variable. Note that these random variables are decidedly *not* independent.

This leaves us with a joint distribution over 6 variables, which we would somehow like to model and fit to a dataset.

---

## slide 22, Probabilistic Methods 1

$$\mathrm{p}(x, y) = \mathrm{p}(x \mid y)\mathrm{p}(y)$$

Remember this rule from the first Probabilistic Methods lecture. If we have a joint distribution over more than two variables on the left, we can apply this rule multiple times.

---

## chain rule *of probability*

$$\begin{aligned}&\mathrm{p}(W_4, W_3, W_2, W_1)\\ &\quad = \mathrm{p}(W_4, W_3, W_2 \mid W_1)\mathrm{p}(W_1)\\ &\quad = \mathrm{p}(W_4, W_3 \mid W_2, W_1)\mathrm{p}(W_2 \mid W_1)\mathrm{p}(W_1)\\ &\quad = \mathrm{p}(W_4 \mid W_3, W_2, W_1)\mathrm{p}(W_3 \mid W_2, W_1)\mathrm{p}(W_2 \mid W_1)\mathrm{p}(W_1)\end{aligned}$$

$$\mathrm{p}(\text{prize} \mid \text{a}, \text{won}, \text{have}, \text{you}, \text{congratulations})$$

16

This gives us the **chain rule of probability** (not to be confused with the chain rule of calculus, which is entirely different), which is often used in modelling sequences.

The chain rule allows us to break a joint distribution on many variables into a product of conditional distributions. In sequences, we often apply it so that each word becomes conditioned on the words before it.

This tells us that if we build a model that can estimate the probability p(x|y, z) of a word x based on the words y, z that precede it, we can then *chain* this estimator to give us the joint probability of the whole sentence x, y, z.

log p(sentence) =

$\Sigma_{word}$ log p(word| all words before word)

17

In other words, we can rewrite the probability of a sentences as the product of the probability of each word, conditioned on its history. If we use the log probability, this becomes a sum.

Note that applying the chain rule in a different order would allow us to condition any word on any other word, but conditioning on the history fits well with the sequential nature of the data, and will allow us to make some useful simplifications later.

## language model

$p(W \mid \text{the, man, fell, out, of, the})$

the man fell out of the …

- window
- aquarium
- pool
- cycling

18

A perfect language model would encompass everything we know about language: the grammar, the idiom and the physical reality it describes. For instance, it would give window a very high probability, since that is a very reasonable way to complete the sentence. Aquarium is less likely, but still physically possible and grammatically correct. A very clever language model might know that falling out of a pool is not physically possible (except under unusual circumstances), so that should get a lower probability, and finally cycling is ungrammatical, so that should get very low probability (perhaps even zero).

The problem is most sentences of this length will never have been seen before in their entirety. A simple way to get a basic model is **to limit how far back we look in the sentence**.

## Markov assumption

$p(\text{prize} \mid \text{a, won, have, you, congratulations}) =$
$p(\text{prize} \mid \text{a, won})$

19

This is called a **Markov assumption**. It's a bit like the naive Bayes assumption: we know it's incorrect, but it still yields a very usable model. The number of words we retain in the conditional is called the order of the Markov model: this is a Markov assumption for a second-order Markov model.

## Markov model

p(prize, a, won, have, you, congratulations)
=p(prize | a, won) p(a | won, have)
p(have | you, congratulations) p(you | congratulations) p(congratulations)

$$p(\text{prize} \mid \text{a, won}) \approx \frac{\#\text{“won a prize”}}{\#\text{“won a”}}$$

2nd *order* Markov model

20

Using the Markov assumption and the chain rule together, we can model a sequence as limited-memory conditional probabilities. These probabilities can then be very simply estimated from a large dataset of text (called **a corpus**).

To estimate the probability of prize given “won a” we just count how often “won a prize” occurs as a proportion of the times “won a” occurs. In other words, how often “won a” is followed by prize.

These n-word snippets are called **n-grams**. “won a prize” is a **trigram**, and “won a” is a **bigram**.

This type of language model is often called a **Markov model**, because of the Markov assumption of limited memory. The size of the memory is referred to as the **order** of the Markov model. The higher the order of your model, the more you can model,

but the more data you'll need for accurate estimates if the probability.

## sequential sampling from a language model

start with a small *seed* sequence $s = [w_1, w_2, w_3]$ of tokens.

**loop**:

Sample next word $w$ according to $p(W = w \mid w_1, w_2, \ldots)$

append $w$ to $s$

21

One interesting thing we can do with a Markov model, is to sample from it, step by step. We start with a seed of a few words, and then work out the probability distribution over the next word, given the last n words. We sample from this distribution, append the sample to our text and repeat the process.

Note that with the Markov assumption, we only need the last n elements of the sequence to work out the probabilities.

**Sequential sampling** is also known as **autoregressive sampling**. In the context of Markov models, the sampling process is often called a **Markov chain**.

## sequential sampling: Markov chain

Go thy
ways, wench; serve them, joining their best friend to ransom straight,
And make him dead, deceased, she's dead, she's good, thou hast thou dost thou dost excuse.
Is thy son to church?

CAPULET Ready to love now
Doth grace that I beseech your high majesty.

SIR WALTER BLUNT, with tears: mine ear. Prithee,
tell her you both of Rome of your will: I
pray you, daughter, he is my soul, he proclaim'd
By Richard that thou dead:
Then, as herbs, grace himself an hour.

source http://www.schmipsum.com/ 22

Here is is a bit of text sampled from a Markov model trained on the works of Shakespeare.

(We're essentially training a very simple probabilistic predictor in the fashion of slide 9, and then sampling from it sequentially)

## sequence classification

$p(\text{spam} \mid \text{“congratulations, you have won a prize”})$

23

Let's now look at a sequence-per-instance example: the spam classification task we saw earlier. We'll see how to approach this with a Markov model.

We are after the probability of the class, given the contents of the message.

## Bayes classifier

$$p(\text{spam} \mid W_1, \ldots, W_n) \propto p(W_1, \ldots, W_n \mid \text{spam}) p(\text{spam})$$



We'll train a generative/Bayes classifier and use Bayes rule to flip around the probabilties.

p(spam) can be estimated as the proportion of spam emails in our data set. For the probability of the message given the class, we'll use our language model.

## conditional on class

$$\begin{aligned} &p(\text{prize}, \text{a}, \text{won}, \text{have}, \text{you}, \text{congratulations} \mid \text{spam}) \\ =&p(\text{prize} \mid \text{a}, \text{won}, \text{spam})\ p(\text{a} \mid \text{won}, \text{have}, \text{spam}) \\ &p(\text{won} \mid \text{have}, \text{you}, \text{spam})\ p(\text{have} \mid \text{you}, \text{congratulations}, \text{spam}) \\ &p(\text{you} \mid \text{congratulations}, \text{spam})\ p(\text{congratulations}, \text{spam}) \end{aligned}$$

$$p(\text{prize} \mid \text{a}, \text{won}, \text{spam}) \approx \frac{\#_{\text{spam}}\text{“won a prize”}}{\#_{\text{spam}}\text{“won a”}}$$



We use the chain rule and the Markov assumption to define the probability that a message occurs (given that it's spam).

We then estimate the different conditional probabilities by computing the relative frequencies of bigrams and trigrams, as before, but we compute them only over the **spam** part of our data.

## algorithm: 2nd-order markov model

**training:**

**for each** class c:

**for** n in 1, 2, 3:

count n-grams in all text belonging to class c

**classification:**

given text $w_1, \ldots, w_m$

$$p(c \mid w_1, \ldots, w_m) \propto p(w_1, \ldots, w_m \mid c) p(c)$$

$$p(w_1, \ldots, w_m \mid c) =$$

$$p(w_1 \mid c)\ p(w_2 \mid w_1, c) \prod_{i \in [3, \ldots, m]} p(w_i \mid w_{i-1}, w_{i-2}, c)$$



## Markov modeling: final comments

- 0-order Markov model: Naive Bayes
  For spam classification higher orders don't improve performance. For other tasks they do.
- Short documents are vastly more likely than long ones
  Doesn't matter for classification. In other settings, conditioning on length may be necessary.
- Laplace smoothing
  Same as before: adapt the estimator by adding pseudo-observations

$$p(\text{prize} \mid \text{a}, \text{won}) \approx \frac{1 + \#\text{“won a prize”}}{|V| + \#\text{“won a”}}$$



V is the vocabulary (the set of all words known to the language model).

As before, we can give the pseudo observations a smaller weight than 1, to have less impact on the estimate.

the man fell out of the …

pool
gazebo
cycling

"congratulations you have won a prize"

p(spam|"congrats, a reward has been awarded to you")

28

These kinds of probability models, and many other models for symbolic sequential data, is that they model symbols as entirely separate objects. They don't exploit the fact that some words have similar meanings or syntactic functions.

For instance, the Markov assumption helps us with the man falling out of the pool, because "out of the pool" is a common phrase, even if nobody falls out of it. However, for rarer cases like "out of the gazebo" vs "out of the cycling" it may be that both trigrams never appear in the data. In that case, we might still say that the first is more likely because it at least ends in a *noun*, which is expected.

Similarly, if our training data give us little information about the message shown here, we would want a model that can conclude that the words in the sentences are very similar to words that do occur in the spam training corpus.

To deal with both cases, we need to somehow model word *similarity*. There are many ways to do this. One popular choice is to use an **embedding model**.

## embedding models

Model object **x** by embedding vector $e_x$.

If $e_x$ and $e_y$ are "similar," so are **x** and **y**

Learn the parameters of $\{e_x\}$ from data.

Cf: latent vectors from autoencoders, PCA, etc.

29

Embedding models take a set of discrete objects, and assign vectors to them. These vectors are then learned from data, based on some loss over the embeddings.

(This is a bit like mapping images to latent vectors in an autoencoder, except we learn the latent representation directly, instead of learning a function from some representation into the late representation).

## distributional hypothesis

Words that occur in the same context often have similar meanings.

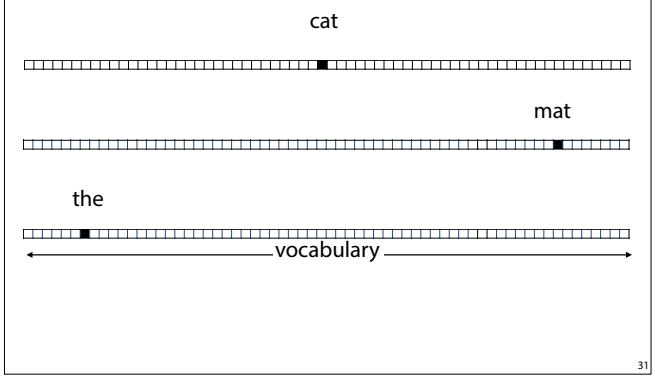
30

The **distributional hypothesis** provides us with a sound way to train these embeddings.

To paraphrase: if we have some word X, and we create a probability distribution P modelling how likely a word Y is to occur in the context of X, we can use that distribution P as a proxy for the *meaning* of X.

The algorithm we will discuss, Word2Vec, provides learns a simple mapping from the embedding to a probability distribution.

## 1-hot vector




31

To start with, we'll represent words as atomic objects: in a single, very large one-hot vector.

These are very big, but we can usually implement things in a clever way so that we won't explicitly need to store these in memory.

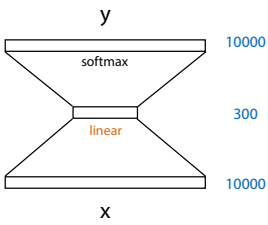
---

## Word2Vec (skipgram version)

shall i compare thee to a summers day thou art more lovely …

x

| x | y |
|---|---|
| compare | shall |
| compare | i |
| compare | thee |
| compare | to |
| thee | i |
| thee | compare |
| thee | to |
| thee | a |
| to | compare |
| to | thee |
| to | a |
| to | summers |
| a | thee |
| a | to |
| a | summers |
| a | day |
| summers | to |




32

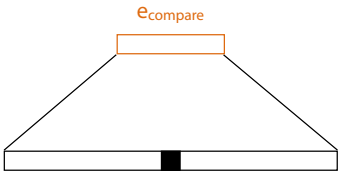
We slide a context window over the sequence. The task is to predict the distribution p(y|x): that predict which words are likely to occur in the context window given the middle word.

We create a dataset of word pairs from the entire text and feed this to a very simple two-layer network. This is a bit like an autoencoder, except we're not reconstructing the output, but predicting the *context*.

The softmax activation over 10k outputs is very expensive to compute, and you need some clever tricks to make this feasible (called *hierarchical* softmax or negative sampling). We won't go into them here.

---

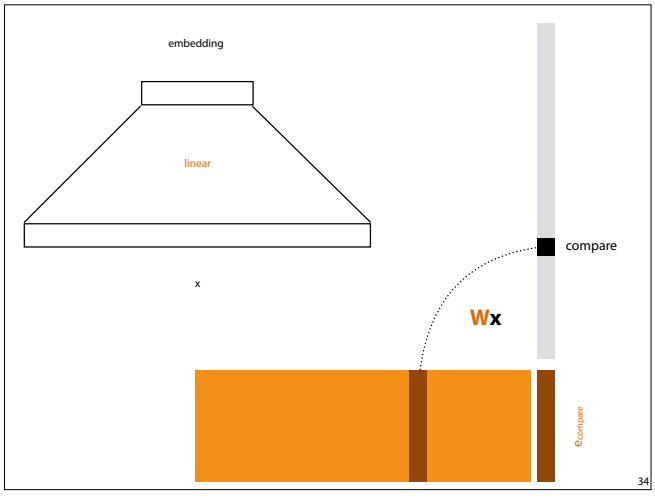
## after training




33

After training, we discard the second layer, and use only the embeddings produced by the first layer.

---



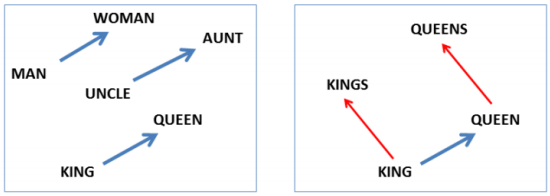

34

Because the input layer is just a matrix multiplication, and the input is just a one-hot vector, what we end up doing when we compute the embedding for word i, is just extracting the i-th column from **W**.

In other words, we're not really training a function that computes an embedding for each word, we are actually learning the embeddings directly: every element of every embedding vector is a separate parameter.

v(king) + v(woman) - v(man) ≈ v(queen)

"feminine" vector




(Mikolov et al., NAACL HLT, 2013)

35

Famously, Word2Vec produces not just an informative embedding, where similar words are close together, but for many concepts, there seems to be a kind of algebraic structure, similar to the smile vector example from the autoencoder.

## word2vec summary

Word2Vec creates **embedding vectors for words**.

In the embedding space distances and directions reflect semantic meaning.

Word2Vec embeddings are a great starting point for deep learning projects on natural language.

Standard W2V embeddings can be downloaded from Google.

36

## break


Garfield generated by a Markov model

source: **https://blog.codinghorror.com/markov-and-you/**

37

## sequences in deep learning

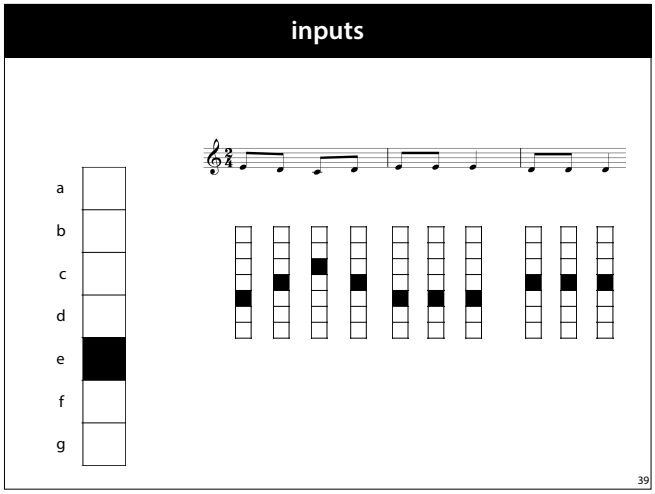
Sequence models: operate on inputs of *different lengths* (using the same weights).

input: *raw* sequence data
deep learning is *end-to-end* learning

output: classificiation, regression, token prediction

layers: sequence-to-sequence

38

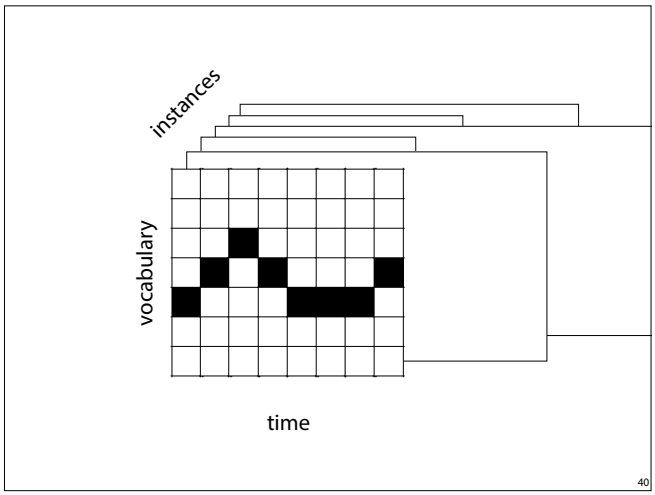



As we've seen, when we want to do deep learning, our input should be represented as a tensor. Preferably in a way that retains all information (i.e. we want to be learning from the raw data, or something as close to it as possible).

Here is an example: to encode a simple monophonic musical sequence, we just one-hot encode the notes, and encode note sequence as a matrix: one dimension for time, one dimension for the notes. We can do the same thing for characters.

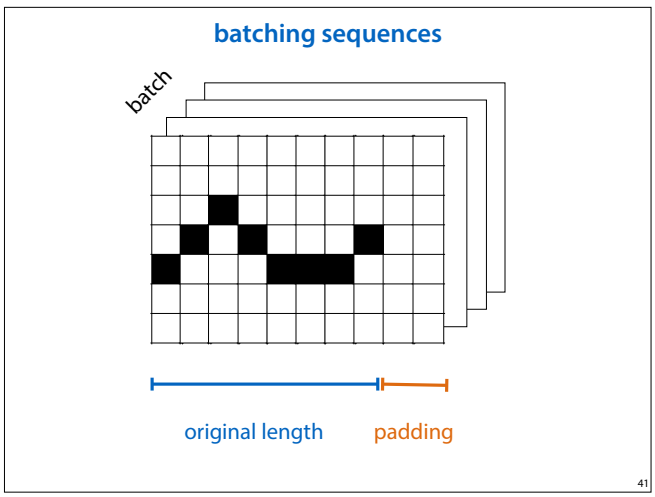
source: **https://violinsheetmusic.org**

---




If we have multiple sequences of different lengths, this leads to a data set of matrices of different sizes.

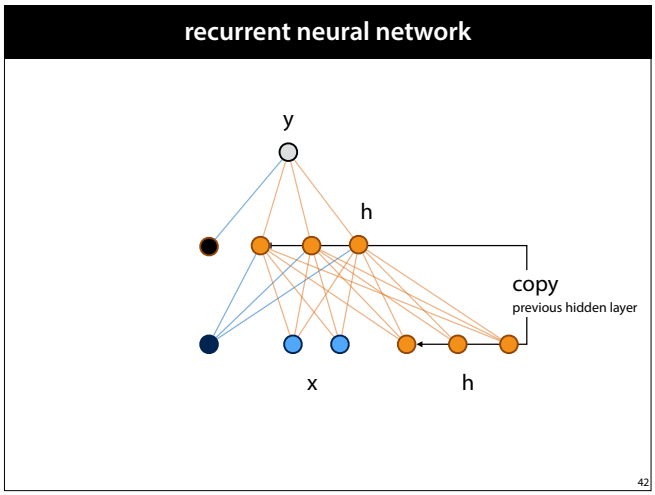
---




In principle, this is not a problem, we want to build models that can deal with sequences of any length (and can generalise over sequences of variable lengths).

However, within a batch it's usually required that all sequences have the same length. Often, we put sequences of similar length together (for instance by sorting the data by length) and then pad the shorter sequences with zero vectors, so that all sequences are the same length.

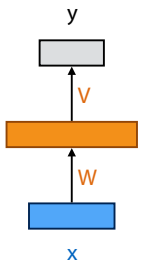
---




One of the most popular neural networks for sequences is the **recurrent neural network**. This is a generic name for any neural network with cycles in it.

The figure shows a popular configuration. It's a basic fully connected network, except that its input x is extended by three nodes to which the hidden layer is copied.

## visual shorthand



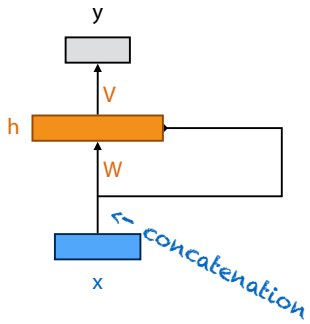

Adapted from
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

43

To keep things clear we will adopt this visual shorthand: a rectangle represents a vector of nodes, and an arrow feeding into such a rectangle annotated with a weight matrix represents a fully connected transformation.

We will assume bias nodes are included without drawing them.

## visual shorthand

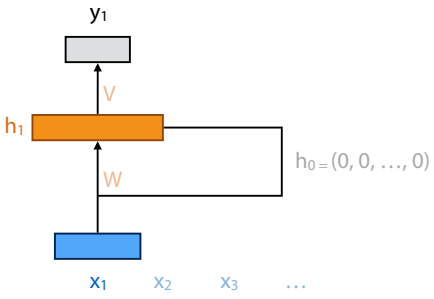



44

A line with no weight matrix represents a copy of the input vector. When two lines flow into each other, we concatenate their vectors.

Here, the added line copies h, concatenates it to x, and applies weight matrix W.

## RNNs on sequences

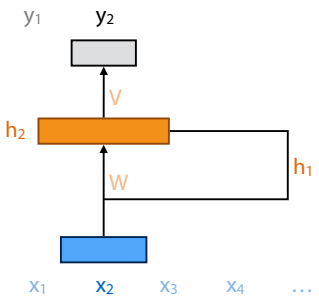



45

We can now apply this neural network to a sequence. We feed it the first input, $x_1$, result in a first value for the hidden layer, $h_1$, and retrieve the first output $y_1$.

The hidden nodes are initialise to zero, so at first the network behaves just like a fully connected network.

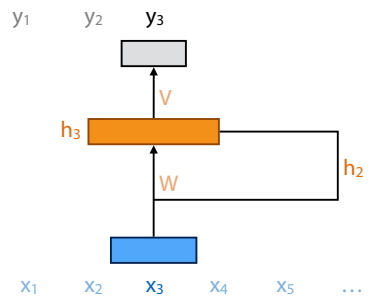
## RNNs on sequences




46

We then feed the second input in the sequence, $x_1$. We now receive the *previous* hidden layer, $h_1$, concatenate it to the input, and multiply it by W, to produce the second hidden layer $h_2$. This is multiplied by V to produce the second output.

## RNNs on sequences




47

And so on.

## how to train RRNs?

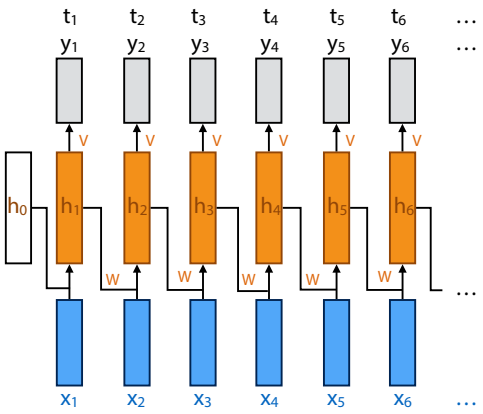
Provide an input sequence **x** and a target sequence **t**.

Backpropagation Through Time (BPTT).

48

In principle, we

## how to train RRNs? unrolling



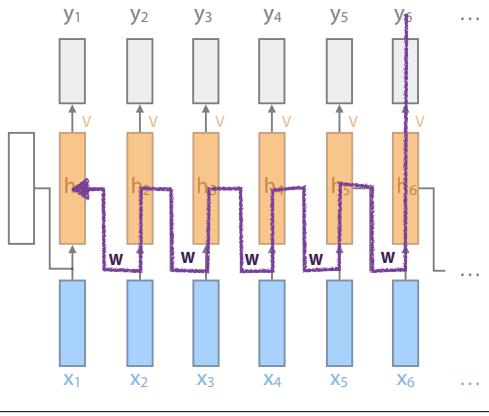

49

Instead of visualising a single small network, applied at every time step, we can **unroll** the network. Every step in the sequence is applied in parallel to a copy of the network, and the recurrent connection flows from the previous copy to the next.

Now the whole network is just one big, complicated **feedforward net**, that is, a network *without* cycles. Note that we have a lot of shared weights, but we know how to deal with those.

The hidden layer is initialised to the zero vector.

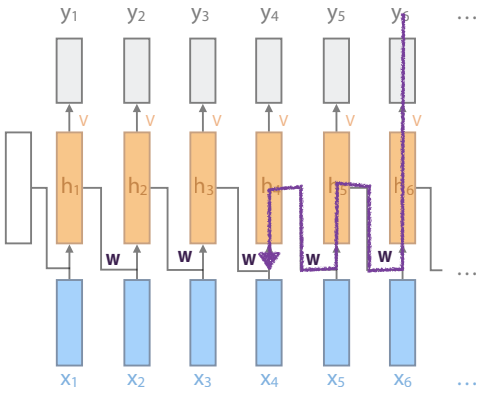
## backpropagation through time




50

Now the whole network is just one big, complicated feedforward net. Note that we have a lot of shared weights, but we know how to deal with those.

## truncated backpropagation through time




51

In *truncated* backpropagation through time, we limit how far back the backpropagation goes, to save memory. The output is still entirely dependent on the whole sequence, but the weights are only trained based on the last few steps. Note that the weights are still affected everywhere, because the are shared between timesteps.

## RNNs: thing to note

sequence-to-sequence model.

no Markov assumption (potentially infinite memory).

fixed set of weights, variable input length.

52

## what can we use RNNs for?

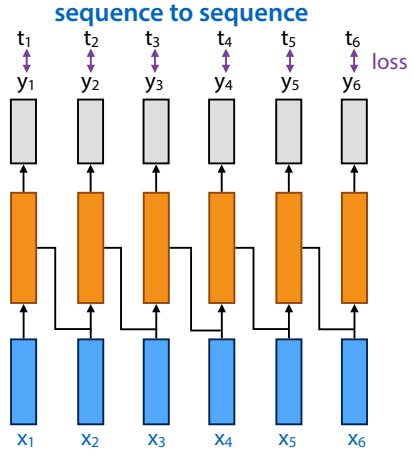
Sequence to sequence:
Eg. Translating English to French

Sequence to label
Eg. sequence classification, regression

Label to sequence
E.g. (conditional) sentence generation

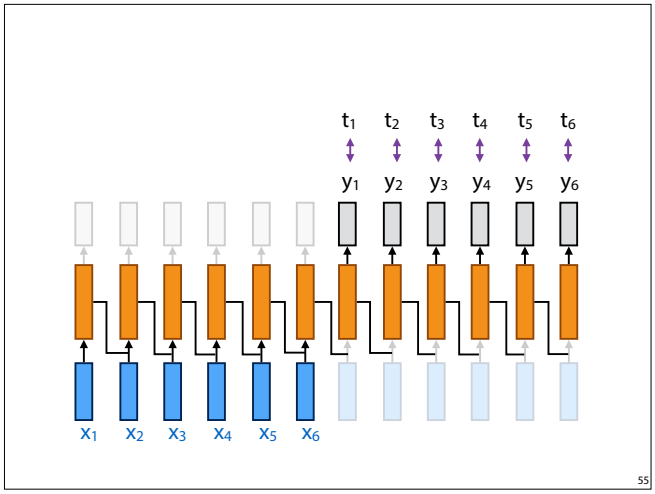
53

## sequence to sequence




54

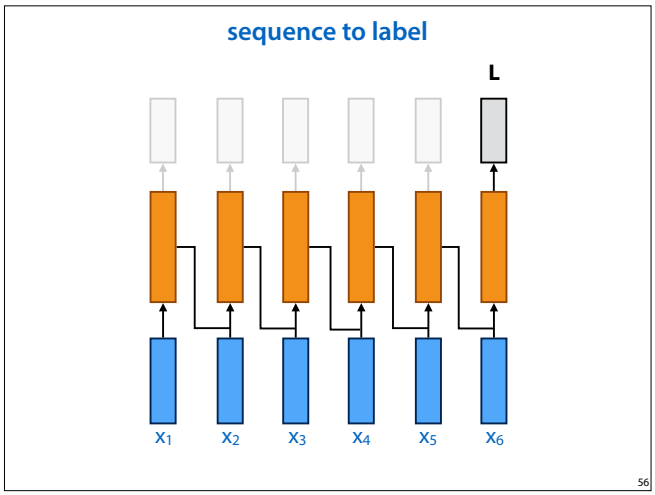
If is the most straightforward way of doing sequence to sequence modeling. Note that the model is now forced to produce the first output before seeing the rest of the input.




If we feed the input and output to the model like this (where the greyed out boxed represent zero vectors), the model gets to read the whole input before having to decide on an output.

Sometimes people even put a fixed number of zero vectors in between the input and the output to allow the model some "time" to process.

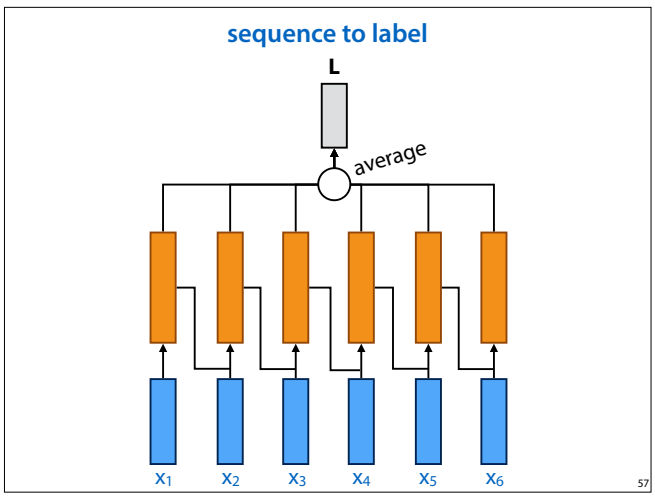
---




If we want to map sequences to labels, we could just ignore all outputs except the last, and take that to be the label. We can compare it to the target label and back propagate the error.

However, this puts a long distance between the first input and the label, and a very short distance between the last input and the label. This asymmetry may mean the model ignores information at the start of the sequence.

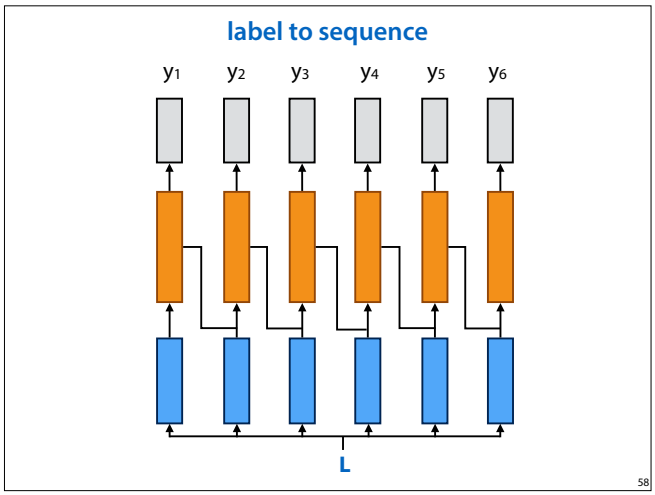
---

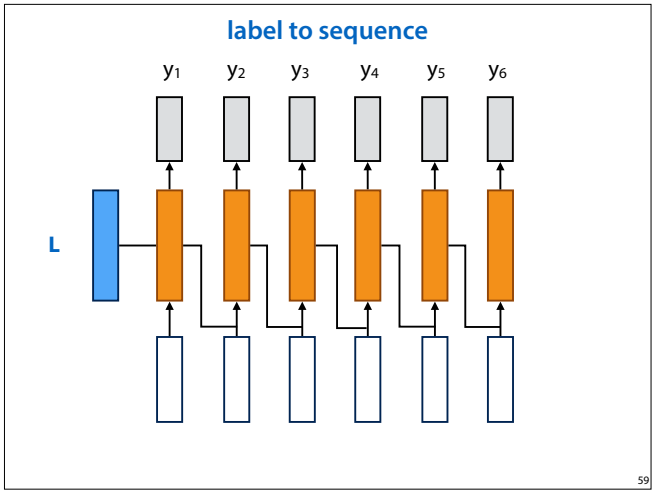



A better approach is to average the outputs over the whole sequence.This makes the distance to the label the same for every input.

Note that the model should be able to handle sequences of variable lengths: this means we can't put a fully connected layer on the top to combine the whole sequence into a single label.

---






We can also use the hidden layer at timestep 0 as an input for the label, and zero the sequence inputs (or use them for something else).

## the problem of long-term dependence

I was born in France, as matter of fact in a little village near Paris, it's famous for its pain-au-chocolat, I lived there until I was 16, when I moved to Amsterdam, so I'm fluent in…

French

Dutch

Aquarium

60

Basic RNNs work pretty well, but they do not learn to remember information for very long. Technically they can, but the gradient vanished too quickly over the timesteps.

You can't have a long term memory for everything. You need to be selective, and you need to learn to select words to be stored for the long term when you first see them.

In order to remember things long term you need to forget many other things.

## LSTM (1997)

Long short-term memory

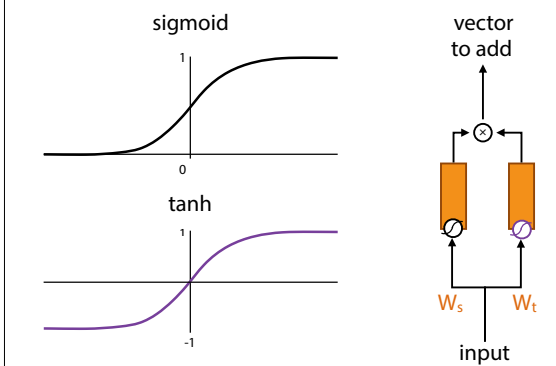
Selective forgetting and remembering, controlled by learnable "gates"

Possibly the first successful deep neural network

61

An enduring solution to the problem are LSTMs. LSTMs have a complex mechanism, which we'll go through step by step, but the main component is a gating mechanism.

## gate




The gate combines the sigmoid and tanh activations. The sigmoid we've seen already. The tanh is just a the sigmoid rescaled so that its outputs are between -1 and 1.

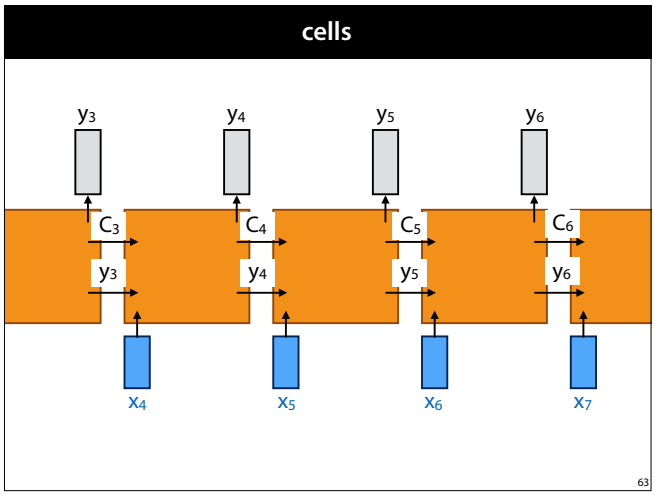
The gating mechanism takes two input vectors, and combines them using a sigmoid and a tanh activation. The gate is best understand as producing an additive value: we want to figure out how much of the input to add to some other vector (if it's import, we want to add most of if, otherwise, we want to forget it, and keep the original value).

The input is first transformed by two weight metrics and then passed though a sigmoid and a tanh. The tanh should be though of as a mapping of the input to the range [-1, 1]. This ensures that the effect of the addition vector can't be too much. The

sigmoid acts as a selection vector. For elements of the input that are important, it outputs 1, retaining all the input in the addition vector. For elements of the input that are not important, it outputs 0, so that they are zeroed out. The sigmoid and tanh vectors are element-wise multiplied.

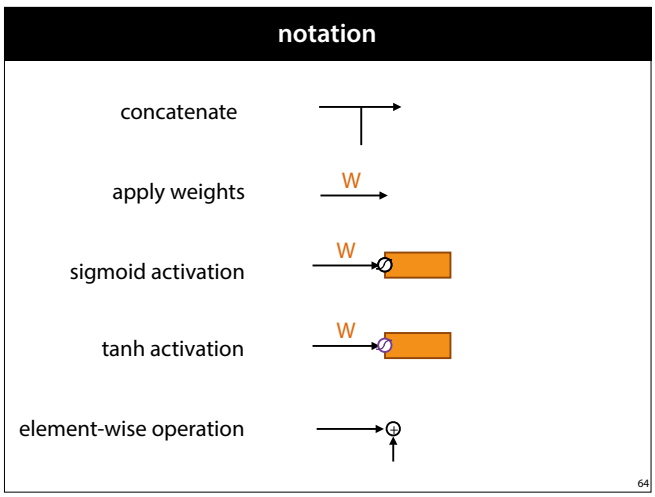
Note that if we initialise Wt and Ws to zero, the input is entirely ignored.

---



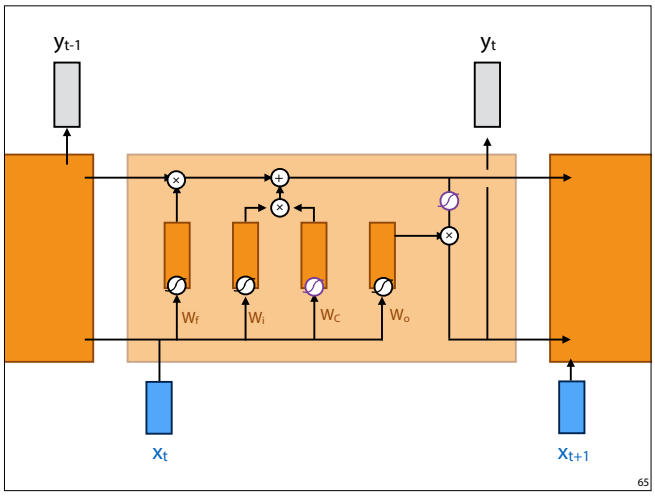

The basic operation of the LSTM is called a cell (the orange square, which we'll detail later). Between cells, there are two recurrent connections, y, the current output, and C the **cell state**.

---

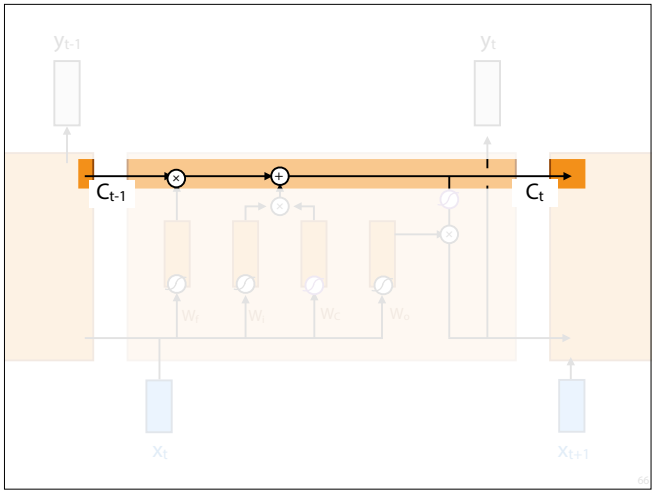



Here is our visual notation.

---



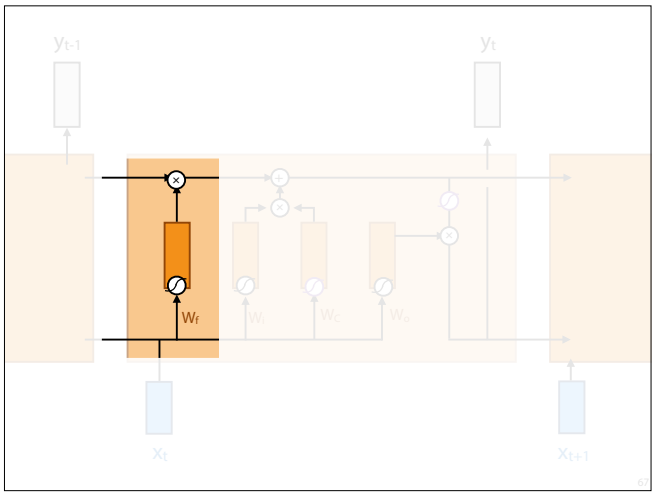

Here is what happens inside the cell. It looks complicated, but we'll go through all the elements step by step.




The first is the “conveyor belt”. It passes the previous cell state to the next cell. Along the way, the current input can be used to manipulate it.

Note that the connection from the previous cell to the next has *no activations*. This means that along this path, gradients do not decay. It’s also very easy for an LSTM cell to ignore the current information and just pass the information along the conveyor belt.

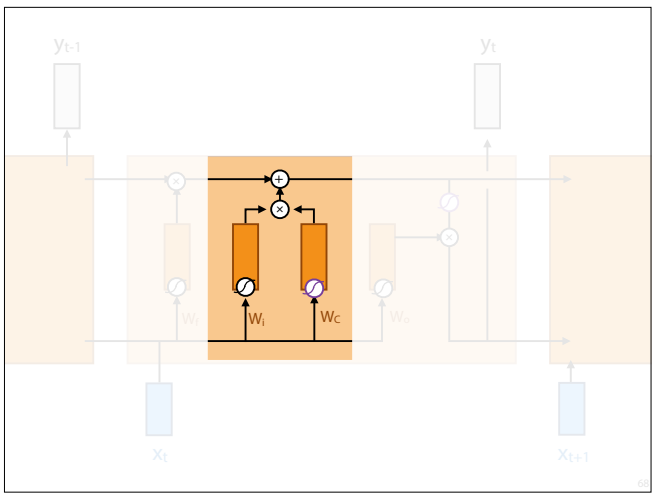
---




Here is the first manipulation of the conveyor belt. This is called the **forget gate**.

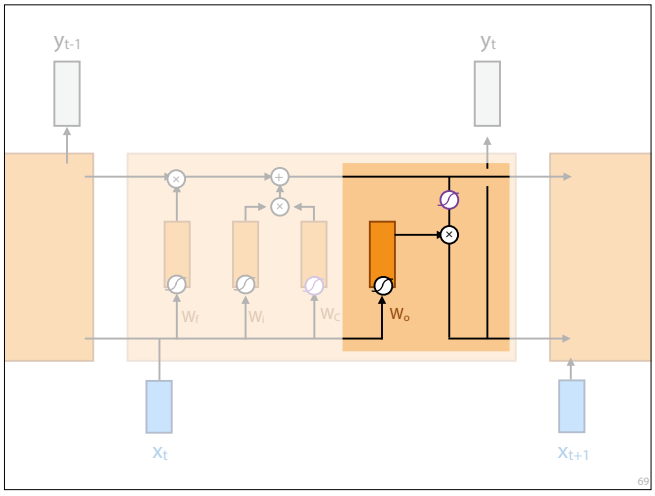
It looks at the current input, concatenated with the previous output, and applies an element-wise scaling to the current value in the conveyor belt. Outputting all 1s will keep the current value on the belt what it is, and outputting all values near 0, will decay the values (forgetting what we’ve seen so far, and allowing it to be replaces by our new values in the next step).

---




in the next step, we pass the input through a generic gate, as described earlier, and add the resulting vector to the value on the conveyor belt.

---




Finally, we need to decide what to output now. We take the current value of the conveyor belt, tanh it to rescale, and element-wise multiply it by another sigmoid activated layer. This layer is sent out as the current output, and sent to the next cell along the second recurrent connection.

## LSTM language model

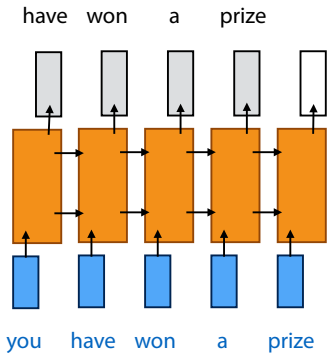
p(prize | a, won, have, you, congratulations)

NB: no Markov assumption!

70

That's a lot of complexity. Let's see what it buys us. The best way, by far, to illustrate the power of LSTMs is to apply the sequential sampling trick.

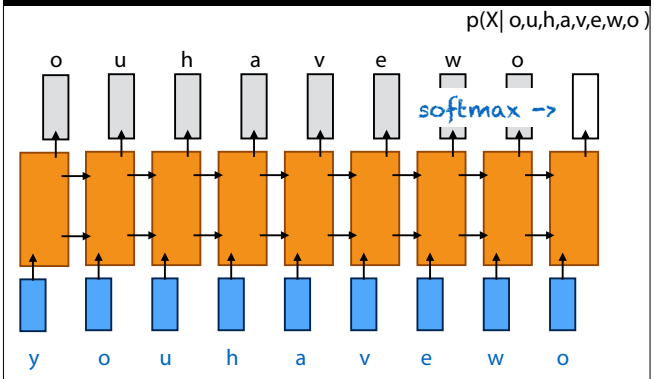
## LSTM language model




71

We train the language model to predict the next token in the sequence. This is easy to do: we just us the sequence shifted by 1 as the training sequence.

## character level language model




72

We'll do this at charter level, to make it more challenging, and we ensure that he network outputs a softmax activated vector over all characters at each timestep. We can then interpret the output of the model at step t as the probability of the character at that point, conditoned on the entire history.

## sequential *sampling* from a language model

start with a small *seed* sequence $s = [c_1, c_2, c_3]$ of tokens.

**loop**:

Sample next char c according to $p(C = c \mid c_1, c_2, \ldots)$
feed the *whole* seed to the network

append c to s

also known as a *autoregressive RNN*

73

Note that this time, as there is no Markov assumption, the network has to see the *whole sequence* so far to predict the next character.

## some examples

source: The Unreasonable Effectiveness of Recurrent Neural Networks
Andrej Karpathy

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

74

## Shakespeare

```
PANDARUS:
Alas, I think he shall be come
approached and the day
When little srain would be
attain'd into being never fed,
And who is but a chain and
subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries,
produced upon my soul,
Breaking and strongly should be
buried, when I perish
The earth and thoughts of many
```

75

Remember, this is a **character level** language model.

## Wikipedia

```
Naturalism and decision for the majority of Arab
countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An
Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were
the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be
said to be directly in Cantonese
Communication, which followed a ceremony and set
inspired prison, training. The
emperor travelled back to [[Antioch, Perth,
October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the
[[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the
conquest of India with the conflict.
Copyright was the succession of independence in
the slop of Syrian influence that
was a famous German movement based on a more
```

76

Note that not only is the language natural, the wikipedia markup is also correct (link brackets are closed properly, and contain key concepts).

```
October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the
[[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the
conquest of India with the conflict.
Copyright was the succession of independence in
the slop of Syrian influence that
was a famous German movement based on a more
popular servicious, non-doctrinal
and sexual power post. Many governments recognize
the military housing of the
[[Civil Liberalization and Infantry Resolution
265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab
Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics
Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those
Socialism's rule,
was starting to signing a major tripad of aid
exile.]]
```

77

The network can even learn to generate valid (looking) URLs for external links.

```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2002-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT
[[Christianity]]</text>
  </revision>
</page>
```

78

Sometimes wikipedia text contains bits of XML for structured information. The model can generate these flawlessly.

---

## LaTeX

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_n} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on $X$, $U$ is a closed immersion of $S$, then $U \to T$ is a separated algebraic space.

*Proof.* Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariosly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps $M$ along the set of points $Sch_{fppf}$ and $U \to U$ is the fibre category of $S$ in $U$ in Section, ?? and the fact that any $U$ affine, see Morphisms, Lemma ??. Hence we obtain a scheme $S$ and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that $f_i$ is of finite presentation over $S$. We claim that $\mathcal{O}_{X,x'}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{V,v'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^* = \mathcal{I}^* \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf} \cdot (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of $X$. Thus $U$ is affine. This is a continuous map of $X$ is the inverse, the groupoid scheme $S$.

*Proof.* See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace $S$ by $X_{spaces,étale}$ which gives an open subspace of $X$ and $T$ equal to $S_{Zar}$, see Descent, Lemma ??. Namely, by Lemma ?? we see that $R$ is geometrically regular over $S$.

**Lemma 0.1.** *Assume (3) and (3) by the construction in the description.*

*Suppose* $X = \lim |X|$ *(by the formal open covering* $X$ *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* $U$ *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_{X'}}).$$

*When in this case of to show that* $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* $T$ *is surjective we may assume that* $T$ *is connected with residue fields of* $S$*. Moreover there exists a closed subspace* $Z \subset X$ *of* $X$ *where* $U$ *in* $X'$ *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) $f$ *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

*Proof.* This is form all sheaves of sheaves on $X$. But given a scheme $U$ and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme $X$ over $S$ at the schemes $X_i \to X$ and $U = \lim X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\ldots,0}$.

**Lemma 0.2.** *Let* $X$ *be a locally Noetherian scheme over* $S$, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \bar{A}_2$ *works.*

**Lemma 0.3.** *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

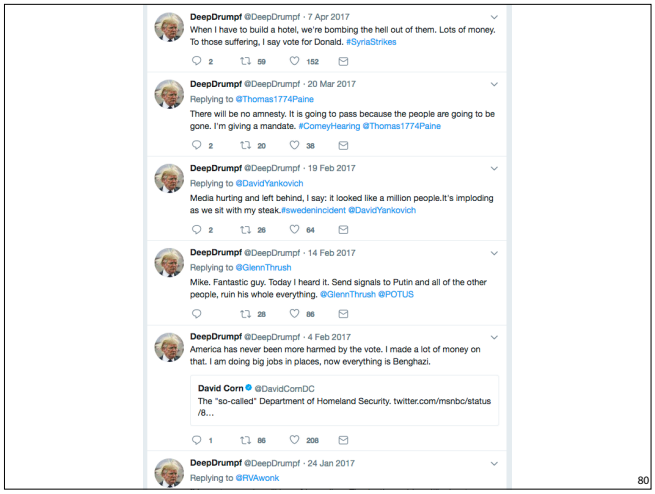
*Proof.* We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{Y'}) = \mathcal{O}_X(D)$$

where $K$ is an $F$-algebra where $\delta_{n+1}$ is a scheme over $S$. □

79

---

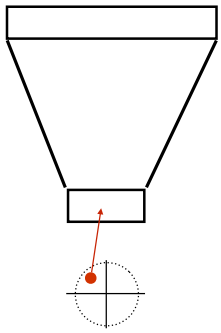



80

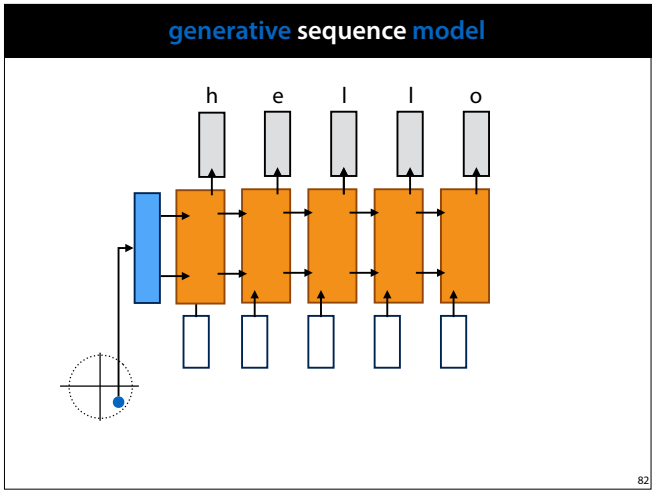
**https://twitter.com/deepdrumpf?lang=en**

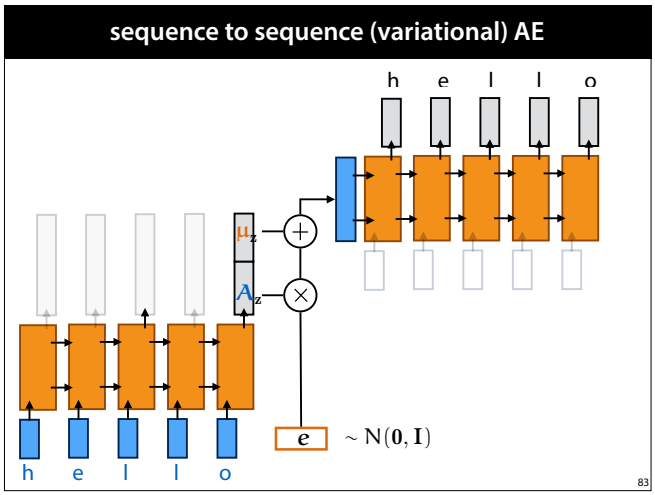
---

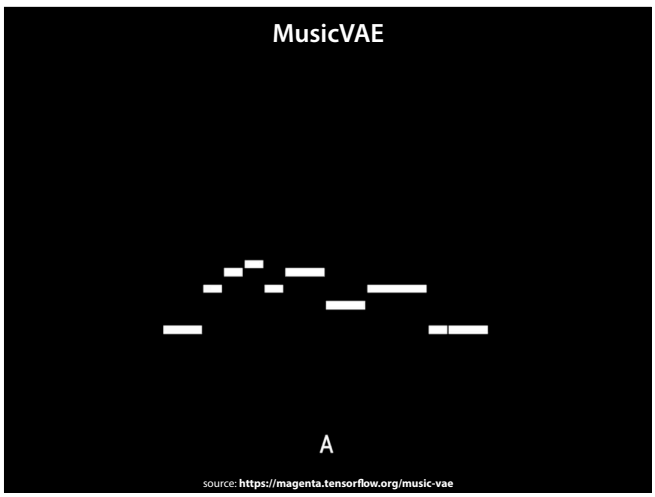
## generative sequence model



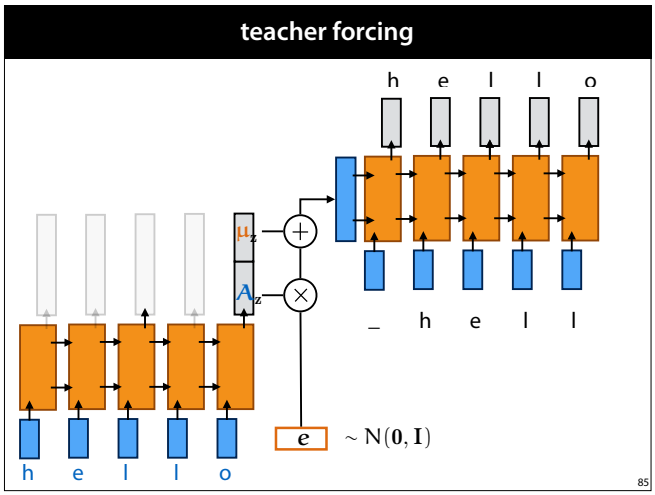
81




source: **https://magenta.tensorflow.org/music-vae**



## sketch-rnn

source: **A Neural Representation of Sketch Drawings**, Ha and Eck (2017)

86

Here we see two interpolation grids for a sequence-to-sequence VAE trained on a a dataset of millions of quickly drawn sketches.

---

## interpolation

**AE**

**i went to the store to buy some groceries .**
*i store to buy some groceries .*
*i were to buy any groceries .*
*horses are to buy any groceries .*
*horses are to buy any animal .*
*horses the favorite any animal .*
*horses the favorite favorite animal .*
**horses are my favorite animal .**

**VAE**

**he was silent for a long moment .**
*he was silent for a moment .*
*it was quiet for a moment .*
*it was dark and cold .*
*there was a pause .*
**it was my turn .**

source: Generating Sentences from a Continuous Space by Samuel R. Bowman

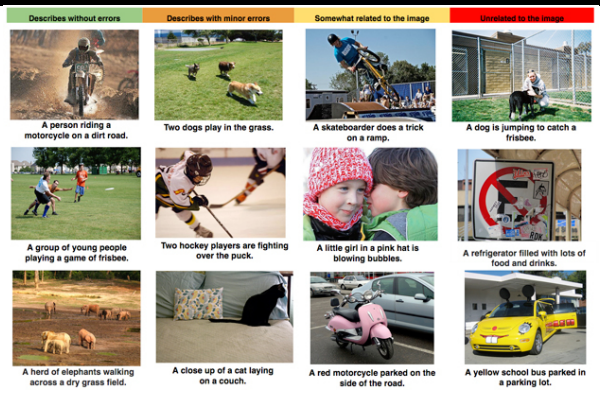
87

Here is an example of *interpolation* on sentences. First using a regular autoencoder, and then using a VAE. Note that the intermediate sentences for the AE are non-grammatical, but the intermediate sentences for the VAE are all grammatical.

*source: Generating Sentences from a Continuous Space by Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, Samy Bengio*
*https://arxiv.org/abs/1511.06349*

---

## natural language descriptions (2014)

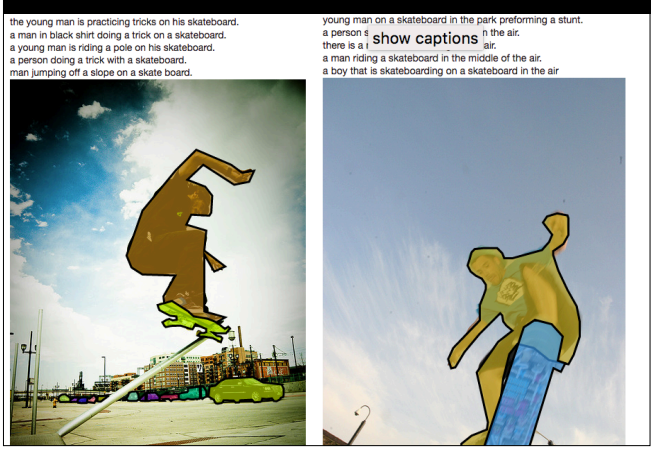



88

*source: https://www.engadget.com/2014/11/18/google-natural-language-image-description/*
*recommended: https://twitter.com/picdescbot*

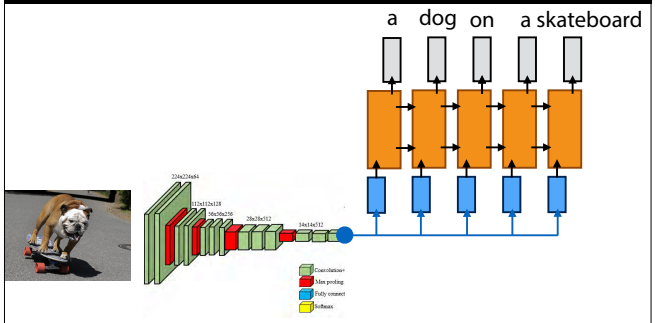
---

## MSCOCO




**cocodataset.org**

## automatic captioning




90

Here's a simple approach to the image captioning problem, which you . We take a pretrained image classification network (see the transfer learning slid from Deep Learning 2) and feed the output to an LSTM. We train end-to-end and use sequential sampling to generate outputs. This is model that you **should be able to train on a high end laptop with a basic GPU.**

This will give you a pretty decently performing model. To reach the state of the art, you also need something called an "attention mechanism." But that's outside the scope of this lecture.

---

## final notes

Word-level LSTMs: use embeddings on the input layer
Either pre-trained w2v embeddings, or an embedding layer that is trained end-to-end

Other, similar architecture: GRUs

Bidirectional RNNs
Two RNNs, in both directions, concatenate the outputs.

91

---

## summary

Sequence modeling: we can use existing methods through feature extraction, but be careful about validation.

Markov models: simple, finite-memory sequence modeling
For classification or generation

Word2Vec: word embeddings reflecting similarity, semantics

LSTMs: incredibly powerful language models. Tricky to train, very opaque.

92

---

**mlcourse@peterbloem.nl**